On Approximation Algorithms for Data Mining Applications

Foto N. Afrati*
National Technical University of Athens, Greece

September 3, 2002

Abstract

We aim to present current trends in the theoretical computer science research on topics which have applications in data mining. We briefly describe data mining tasks in various application contexts. We give an overview of some of the questions and algorithmic issues that are of concern when mining huge amounts of data that do not fit in main memory.

1 Introduction

Data mining is about extracting useful information from massive data e.g., finding frequently occurring patterns or finding similar regions or clustering the data. From the algorithmic point of view mining algorithms seek to compute good approximate solutions to the problem at hand. As a consequence of the huge size of the input, algorithms are usually restricted to making only a few passes over the data, and they have limitations on the random access memory they use and the time spent per data item. The advent of the internet has added new applications and challenges to this area too.

The input in a data mining task can be viewed, in most cases, as a two dimensional $m \times n$ 0,1-matrix which, in many cases, might be sparse. This matrix may represent several objects such as a collection of documents (each row is a document and each column is a word and there is a 1 entry if the word appears in this document), or a collection of retail records (each row is a transaction record and each column represents an item, there is a 1 entry if the item was bought in this transaction), or both rows and columns are sites on the web and there is a 1 entry if there is a link from the one site to the other. In the latter case, the matrix is often viewed as a graph too. Sometimes the matrix can be viewed as a sequence of vectors (its rows) or even a sequence of vectors with integer values (not only 0,1).

*This work was partially supported by European Commission, Project APOLL-IST-1999-14084 and the Greek General Secretariat of Research and Technology, Project on Data Mining

The goal of a *data mining task* is to extract useful (and succinct) information about the data such as *cluster* the rows of the matrix according to various criteria (e.g., similarity), or find association rules between the entries of rows (e.g., if the i'th entry of a row v is 1 then most probably the j-th entry of v is also 1), or *rank the pages* of the web according to their "importance".

The performance of a data mining algorithm is measured in terms of the number of passes and the required work space in main memory. A constant number of passes is acceptable but one pass algorithms are mostly sought for. The workspace available ideally is constant but sublinear space algorithms are also considered. The quality of the output is usually measured using conventional approximation ratio measures [86], although in some problems the notion of approximation and the manner of evaluating the results remain to be further investigated. Space and time per item is measured as a function of N, the number of items read by the algorithm. Since the amount of data far exceeds the amount of workspace available to the algorithm, it is not possible for the algorithm to "remember" large amounts of past data. These constraints call for designing novel techniques and novel computational paradigms. For an example, a commonly used approach is to create a *summary* of the past data to store in main memory, leaving also enough memory for the processing of the future data. Using a random *sample* of the data is also another popular technique. Several single pass algorithms have been developed recently and also techniques and tools that facilitate them and we will review some of them.

Besides data mining, other applications can be also modeled as one pass problems such as the interface between the storage manager and the application layer of a database system or processing data that are brought to desktop from networks, where each pass essentially is another expensive access to the network. Several communities have contributed (with technical tools and methods as well as by solving similar problems) to the evolving of the data mining field, including statistics, machine learning and databases.

The rest of this paper is organized as follows. In the next section, we provide some theoretical background restricting ourselves to providing a formal presentation and motivation of some of the most typical research directions. In section 3, we list some of the most important data mining tasks and mention some of its applications. In section 4, we present recent algorithms for solving a distance approximation problem arising in several mining tasks. Then in each of the other sections, we present research done and in progress for each task. Technical tools and methodologies even modules of algorithms may be thought of as been of use to various tasks. However many algorithms proven efficient for practical applications are tailored to a specific task. The exposition here is meant to serve as an introduction to algorithms for data mining applications. Its goal is to hint on issues that abide with data mining tasks and theoretical computer science.

2 Technical Background

In this section we present some theoretical results and formalisms that are often used in developing algorithms for data mining applications. In this context, the singular value decomposition (SVD) of a matrix (subsection 2.1.2) has inspired web search techniques, and, as a dimensionality reduction technique, is used for finding similarities among documents or clustering documents (known as the latent semantic indexing technique for document analysis). Random projections (subsection 2.1.1) offer another means for dimensionality reduction explored in recent work. Data streams (subsection 2.2) is proposed for modeling limited pass algorithms; in this subsection some discussion is done on lower and upper bounds on the required workspace. Sampling techniques have also been used in statistics and learning theory, under somewhat different perspective however. Storing a sample of the data that fits in main memory and running a "conventional" algorithm on this sample is often used as the first stage of various data mining algorithms. We discuss (subsection 2.3) the problem of selecting a random sample out of a dataset. We present a computational model for probabilistic sampling algorithms that compute approximate solutions. This model is based on the decision tree model [25] and relates the query complexity to the size of the sample.

We start by providing some (mostly) textbook definitions for self containment purposes. In data mining we are interested in vectors and their relationships under several distance measures. For two vectors, $\vec{v} = (v_1, \ldots, v_n)$, $\vec{u} = (u_1, \ldots, u_n)$, the *dot product* or *inner product* is defined to be a number which is equal to the sum of the component-wise products $\vec{v} \cdot \vec{u} = v_1 u_1 + \ldots + v_n u_n$ and the L_p *distance* (or L_p *norm*) is defined to be: $||\vec{v} - \vec{u}||_p = (\Sigma_{i=1}^n |v_i - u_i|^p)^{1/p}$. The L_p distance is extended to be defined between matrices : $||\vec{V} - \vec{U}||_p = (\Sigma_i (\Sigma_j |V_{ij} - U_{ij}|^p))^{1/p}$. We sometimes use $|| \;\; ||$ to denote $|| \;\; ||_2$. The *cosine distance* is defined to be $1 - \frac{\vec{v} \cdot \vec{u}}{||\vec{v}|| \; ||\vec{u}||}$. For sparse matrices the cosine distance is a suitable similarity measure as the dot product deals only with non-zero entries (which are the entries that contain the information) and then it is normalized over the lengths of the vectors.

Given a sequence of reals $x_1, \ldots, x_n$, the *k-th statistical moment* is defined as $\mu_k(x_1, \ldots, x_n) = 1/n \Sigma_{i=1}^n x_i^k$. The first statistical moment is simply the *mean*.

For definitions on random variables and their distributions that may be needed in the following sections refer to [8]. Some results are based on *stable distributions* [74]. A distribution $\mathcal{D}$ over the reals is called *p-stable* if for any n real numbers $a_1, \ldots, a_n$ and independent identically distributed, with distribution $\mathcal{D}$, variables $X_1, \ldots, X_n$, the random variable $\Sigma_i a_i X_i$ has the same distribution as the variable $(\Sigma_i |a_i|^p)^{1/p} X$, where X is a random variable with the same distribution as the variables $X_1, \ldots, X_n$. It is known that stable distributions exist for any $p \in (0, 2]$.

A *randomized algorithm* [70] is an algorithm that flips coins, i.e., it uses random bits, while no probabilistic assumption is made on the distribution of the input. A randomized algorithm is called *Las-Vegas* if it gives the correct answer on all inputs. Its running time or workspace could be a random variable depending on the random variable of the coin tosses. A randomized algorithm is called *Monte-Carlo with error probability* ϵ if on every input it gives the right answer with probability at least $1 - \epsilon$.

2.1 Dimensionality Reduction

Given a set S of objects and an object A in S, consider the problem of finding those objects that are "similar" to A. Each object is represented by an n-dimensional tuple and for checking similarity, we assume some distance metric, usually where the triangle inequality holds.

A naive way to attack the problem is indexing the tuples by using a high dimensional index structure (such as R-trees, grid-files). High dimensional index structures partition the space or dataset into regions and search only those regions that potentially contain objects "similar" to A. This is problematic because in high dimensions, the distance computations are very slow and moreover the index structure performance is decreased (known as the Curse of Dimensionality).

Thus, embedding a data set of points into a space of low dimensions preserving interesting properties is an issue. Several data mining tasks benefit using an efficient algorithm that solve variants of this problem:

Problem (Dimensionality reduction): Given a set S of m points in the d-dimensional space over the reals, find a small k such that there is a mapping from S onto a set of m points in the k-dimensional space over the reals, which approximately preserves distances.

Dimensionality reduction techniques that are popular recently include Random Projections and Singular Value Decomposition (SVD). Other dimensionality reduction techniques have used the first few components of the Discrete Fourier Transform (DFT) (a heuristic which is based on the observation that, for many sequences, most of the energy of the signal is concentrated in those components), or other linear transformations (Discrete Cosine transform or Haar Wavelet coefficients). The L_2 distance is preserved exactly under the DFT and it is practically efficient due to an $O(nlogn)$ DFT algorithm. Dimensionality reduction techniques are well explored in databases.

It is worth noticing, in this context, that in high dimensions counter-intuitive facts hold. For example, for any pair of points, the distance (under the most common distance measures) is equal to average with high probability. Also, they are orthogonal with high probability.

2.1.1 Random Projections

The Johnson-Lindenstrauss (JL) lemma [57] states that any set of n points can be embedded into the k-dimensional space with $k = O(\log n/\epsilon^2)$ so that the distances are preserved within a factor of ϵ.

Lemma 2.1 *(JL) Let $\vec{v}_1, \ldots, \vec{v}_m$ be a sequence of points in the d-dimensional space over the reals and let $\epsilon, F \in (0,1]$. Then there exists a linear mapping f from the points of the d-dimensional space into the points of the k-dimensional space where $k = O(\log(1/F)/\epsilon^2)$ such that the number of vectors which approximately preserve their length is at least $(1-F)m$. We say that a vector $\vec{v}_i$ approximately preserves its length if:*

$$k||\vec{v}_i||^2 \leq ||f(\vec{v}_i)||^2 \leq k(1+\epsilon)||\vec{v}_i||^2$$

□

The proof of the lemma, though, is non-constructive: it shows that a random mapping induces small distortions with high probability. Several versions of the proof exist in the literature. We sketch the proof from [55]. Since the mapping is linear, we can assume without loss of generality that the $\vec{v}_i$'s are unit vectors. The linear mapping f is given by a $k \times d$ matrix $\vec{A}$ and $f(\vec{v}_i) = \vec{A}\vec{v}_i, i = 1, \ldots, m$ By choosing the matrix $\vec{A}$ at random such that each of its coordinates is chosen independently from $N(0,1)$, then each coordinate of $f(\vec{v}_i)$ is also distributed according to $N(0,1)$ (this is a consequence of the spherical symmetry of the normal distribution). Therefore, for any vector $\vec{v}$, for each $j = 1, \ldots, k/2$, the sum of squares of consecutive coordinates $Y_j = ||f(\vec{v})_{2j-1}||^2 + ||f(\vec{v})_{2j}||^2$ has exponential distribution with exponent 1/2. The expectation of $L = ||f(\vec{v})||^2$ is equal to $\Sigma_j E[Y_j] = k$. It can be shown that the value of L lies within ϵ of its mean with probability $1 - F$. Thus the expected number of vectors whose length is approximately preserved is $(1 - F)m$. The JL lemma has been proven useful in improving substantially many approximation algorithms (e.g., [55, 15]). Recently in [37], a deterministic algorithm is presented which finds such mapping in time almost linear in the number of distances to preserve times the dimension d of the original space. In recent work, random projections are used to compute summaries of past data called *sketches* to solve problems such as approximating the L_p norm of a data stream.

2.1.2 Singular Value Decomposition

Consider matrices with real numbers as entries. We say that a matrix $\vec{M}$ is *orthogonal* if $\vec{M}\vec{M}^{Tr} = \vec{I}$ where $\vec{I}$ is the identity matrix (for any matrix $\vec{A}$, $\vec{A}^{Tr}$ denotes the transpose of matrix $\vec{A}$). An *eigenvalue* of a $n \times n$ matrix $\vec{M}$ is a number λ such that there is a vector $\vec{t}$ which satisfies $\vec{M}\vec{t} = \lambda\vec{t}$. Such a vector $\vec{t}$ is called an *eigenvector* associated with λ. The set of all eigenvectors associated with λ form a subspace and the dimension of this subspace is called the *multiplicity* of λ. If $\vec{M}$ is a symmetric matrix, then the multiplicities of all eigenvalues sum up to n. Let us denote all the eigenvalues of such a matrix $\vec{M}$ by $\lambda_1(\vec{M}), \lambda_2(\vec{M}), \ldots \lambda_n(\vec{M})$, where we have listed each eigenvalue a number of times equal to its multiplicity. For symmetric matrix $\vec{M}$, we can choose for each $\lambda_i(\vec{M})$ an associated eigenvector $\vec{t_i}(\vec{M})$ such that the set of vectors $\{\vec{t_i}(\vec{M})\}$ forms an orthonormal basis for the n-dimensional space over the real numbers. Let $\vec{Q}$ be the matrix with columns these vectors and let Λ be the diagonal matrix with diagonal entries the list of eigenvalues. Then, it is easy to prove that: $\vec{M} = \vec{Q}\Lambda\vec{Q}^{Tr}$. However the result extends to any matrix as the following theorem states.

Theorem 2.1 *(Singular Value Decomposition/SVD) Every $m \times n$ matrix $\vec{A}$ can be written as $\vec{A} = \vec{U}\vec{T}\vec{V}^{Tr}$ where $\vec{U}$ and $\vec{V}$ are orthogonal and $\vec{T}$ is diagonal.* □

The diagonal entries of $\vec{T}$ are called the *singular values* of $\vec{A}$. It is easy to verify that the columns of $\vec{U}$ and $\vec{V}$ represent the eigenvectors of $\vec{A}\vec{A}^{Tr}$ and $\vec{A}^{Tr}\vec{A}$ respectively and the diagonal entries of $\vec{T}^2$ represent their common set of eigenvalues. The importance of the SVD in dimensionality reduction lies in the following theorem which states that $\vec{U}$, $\vec{T}$, $\vec{V}$ can be used to compute, for any k, the matrix A_k of rank k which is "closest" to $\vec{A}$ over all matrices of rank k.

Theorem 2.2 *Let the SVD of $\vec{A}$ be given by $\vec{A} = \vec{U}\vec{T}\vec{V}^{Tr}$. Suppose $\tau_1, \ldots, \tau_k$ are the k largest singular values. Let $\vec{u}_i$ be the i-th column of $\vec{U}$ and $\vec{v}_i$ be the i-th column of $\vec{V}$ and let τ_i be the i-th element in the diagonal of $\vec{T}$. Let r be the rank of $\vec{A}$ and let $k < r$. If*

$$\vec{A}_k = \Sigma_{i=1}^k \tau_i \vec{u}_i \vec{v}_i^{Tr}$$

Then

$$\mathbf{min}_{rank(\vec{B})=k} ||\vec{A} - \vec{B}||_2 = ||\vec{A} - \vec{A}_k||_2 = \tau_{k+1}$$

□

The SVD technique displays optimal dimensionality reduction (for linear projections) but it is hard to compute.

2.2 The Data Stream Computation Model

The streaming model is developed to formalize a single (or few) pass(es) algorithm over massive data that do not fit in main memory. In the stream model, the data is observed once (or few times) and in the same order it is generated. We seek to minimize the required workspace and the time to process each data item. Streaming models have been studied recently and methods have been developed for comparing data streams under various L_p distances, or clustering them. The stream model from the database perspective is investigated in the Stanford stream data management Project [82] (see [10] for an overview and algorithmic considerations).

In the interesting work of [52] where the stream model was formalized, a *data stream* is defined as a sequence of data items $v_1, v_2, \ldots, v_n$ which are assumed to be read by an algorithm only once (or very few times) in increasing order of the indices i. The number P of *passes* over the data stream and the *workspace* W (in bits) required by the algorithm in the main memory are measured. The performance of an algorithm is measured by the number of passes the algorithm makes over the data and the required workspace, along with other measures such as the computation time per input data item. This model does not necessarily require a bound on the computation time.

It is natural to ask whether under the stream model there are noticeable differences regarding the workspace requirements (i) between one-pass and multi-pass algorithms, (ii) between deterministic and randomized algorithms and (iii) between exact and approximation algorithms. These questions were explored in [71] and it was shown that: (i) Some problems require a large space in one pass and a small space in two passes. (ii) There can be an exponential gap in space bounds between Monte-Carlo and Las-Vegas algorithms. (iii) For some problems, an algorithm for an approximate solution, requires substantially less space than an exact solution algorithm.

In [52], the lower bounds on the workspace of limited-pass algorithms are shown using tools from the communication complexity area. The space complexity of estimating the frequency moments of a sequence of elements in one pass was studied in [7], where communication complexity was also used to prove almost tight lower bounds on the space requirements.

Communication complexity [68] is defined as follows. In the *(2-party) communication model* there are two players A and B. Player A is given a x from a finite set X and player B is given a y from a finite set Y. They want to compute a function $f(x, y)$. As player A does not know y and player B does not know x, they need to communicate. They use a protocol to exchange bits. The *communication complexity of a function* f is the minimum over all communication protocols of the maximum over all $x \in X, y \in Y$ of the number of bits that need to be exchanged to compute $f(x, y)$. The protocol can be deterministic, Las-Vegas or Monte-Carlo. If one player is only transmitting and one is only receiving then it is called *one-way communication complexity.* In this case, only the receiver needs to be able to compute f.

To see how communication complexity is related to deriving lower bounds on the space, think of one way communication where player A has the information of the past data and player B has the information of the future data. The communication complexity can be used as a lower bound on the space available to store a "summary" of the past data.

An early work in context similar to data streams is [71]. The problem studied is the space required for selecting the k-th largest out of n elements using at most P passes over the data. An upper bound of $n^{1/P} \log n$ and a lower bound of $n^{1/P}$ is shown, for large enough k.

Recent work on space lower bounds include [79]. A distributed stream model is also proposed in [46] which combines features of both streaming models and communication complexity models. The data stream model appears to be related to other work e.g., on competitive analysis [59], or I/O efficient algorithms [87]. However, it is more restricted in that it requires that a data item can never again be retrieved in main memory after its first pass (if it is a one-pass algorithm). In general, a data item can be accessed so many times as the passes that we assume through the data and in the specified order.

2.3 Sampling

Randomly sampling a few data items of a large data input is another technique for building algorithms that find approximate solutions to mining problems. A basic problem in that context is to computes the size of the sample required to determine whether there are k distinct values in a given dataset, each with a frequency greater than ϵ. The following proposition [75] gives a lower bound on the size of the sample.

Proposition 2.1 *If a dataset D contains $l \geq k$ distinct values of frequency at least ϵ, then a sample of size $s \geq \frac{1}{\epsilon} \log \frac{k}{\delta}$ contains at least k distinct values with probability $> \delta$.* □

To prove, let $a_1, \ldots, a_l$ be the l distinct values of frequencies $p_1, \ldots, p_l$ respectively and, each frequency is at least ϵ. Then the probability our sample missed k of these distinct values is at most $\Sigma_{i=1}^{k}(1-p_i)^s \leq k(1-\epsilon)^s \leq \delta$ by our choice of s.

In a similar context, random sampling from a dataset whose size is unknown, is of interest in many specific applications. The problem is to select a random sample of size n from a dataset of size N when N is unknown. A one-pass *reservoir* algorithm is developed in [88]. A *reservoir* algorithm maintains a sample (reservoir) of data items in main memory and data items may be selected for the reservoir as they are processed. The final random sample will be selected from the sample maintained in the reservoir (hence the size of the sample in the reservoir is larger than n).

Next, we describe a computational model to formalize approximation algorithms that use a sample of the data (sampling algorithms) and then derive lower bounds on the required size of the sample for various problems. An algorithm that uses a sample of the input is formalized in [12] as a *uniform randomized decision tree.* A *randomized decision tree* has two kinds of internal nodes, *query* nodes and *random coin toss* nodes. Leaves are related to output values. On an input $x_1, \ldots, x_n$, the computation of the output is done by following a path from the root to a leaf. On each internal node a decision is made as to which of its children the computation path moves next. In a random coin toss node this decision is based on a coin toss which picks one of the children uniformly at random. A query node v has two labels, an input location (to be queried), and a function which maps a sequence of query locations (the sequence is thought of as the input values queried so far along the path from the root) to one of the children of this node v. The child to which the path moves next is specified by the value of this function. Each leaf is labeled by a function which maps the sequence of query locations read along the path to an output value. The output is the value given by the function on the leaf which is the end point of the computation path.

Note that any input $x_1, \ldots, x_n$ may be associated with several possible paths leading from the root to a leaf, depending on the random choices made in the random coin nodes. These random choices induce a distribution over the paths corresponding to $x_1, \ldots, x_n$.

A *uniform* randomized decision tree is defined as a randomized decision tree with the difference that each query node is not labeled by an input variable. The query in this case is done uniformly at random over the set of input values that are not queried so far along the path from the root. A uniform decision tree can be thought as a sampling algorithm which samples the input uniformally at random and uses only these sample values to decide the output. Thus the number of query nodes along a path from the root to a leaf is related to the size of the sample.

The *expected query complexity of a decision tree T on input $\vec{x} = x_1, \ldots, x_n$* denoted $S^e(T, \vec{x})$, is the expected number of query nodes on paths corresponding to $\vec{x}$. The *worst case query complexity of a tree T on input $\vec{x}$*, denoted $S^w(T, \vec{x})$, is the maximum number of query nodes on paths corresponding to $\vec{x}$. Here the expectation and the maximum are taken over the distribution of paths.

The *expected and worst case complexity of T $S^e(T)$* and $S^w(T)$ are the maximum of $S^e(T, \vec{x})$ and $S^w(T, \vec{x})$, respectively, over all inputs $\vec{x}$ in A^n.

Because of the relation between query complexity and the size of the required sample, a relationship can also be obtained between query complexity and space complexity as defined in the data stream model. Let $\epsilon \geq 0$ be an error parameter, δ $(< 0\delta < 1)$ a confidence parameter, and f a function. A decision tree is said to (ϵ, δ)-*approximate* f if for every input $\vec{x}$ the probability of paths corresponding to $\vec{x}$ that output a values y within a factor of ϵ from the exact solution is at least $1 - \delta$. The (ϵ, δ) *expected query complexity* of f is:

$$S^e_{\epsilon,\delta}(f) = \quad min\{S^e(T) \mid T \quad (\epsilon, \delta) - approximates \quad f\}$$

The *worst case query complexity of a function* f is defined similarly.

The (ϵ, δ) query complexity of a function f can be directly related to the space complexity as defined on data streams. If a function has (ϵ, δ) query complexity $S^e_{\epsilon,\delta}(f)$, then the space required in main memory is at most $S^w_{\epsilon,\delta}(f)O(\log |A| + \log n)$, where A, n are parameters of the input vector $\vec{x}$. For input vector $\vec{x} = x_1, \ldots, x_n$, n is the number of data items and A is the number of elements from which the values of each x_i is drawn.

Based on this formalization, a lower bound is obtained on the number of samples required to distinguish between two distributions [12]. It it also shown that the k-th statistical moment can be approximated within an additive error of ϵ by using a random sample of size $O(1/\epsilon^2 \log \frac{1}{\delta})$, and that this is a lower bound on the size of the sample. Work that also refer to lower bounds on query complexity for approximate solutions include results on the approximation of the mean [26], [34, 80], the approximation on the frequency moment [29].

Lossy compression may be related to sampling. When we have files in compressed form, we might want to compute functions of the uncompressed file without having to decompress. Compressed files might even been thought of as not been able to be precisely retreived by decompression, namely the compression (in order to gain larger compression factors) allowed for some loss of information (*lossy compression*). Problems of this nature are related to sampling algorithms in [42].

Statistical decision theory and statistical learning theory are fields where sampling methods are used too. However they focus on different issues than data mining does. Statistical decision theory [14] studies the process of making decisions based on information gained by computing various parameters of a sample. However the sample is assumed given and methods are sought that maximize the utitily of it. Computing the required size of a sample for approximately computing parameters of the input data is not one of its concerns. Statistical learning theory [85, 60] is concerned with learning an unknown function from a class of target functions, i.e., approximating the function rather, whereas, in data mining, the interest is in approximating some parameter of the function.

3 Data Mining Tasks and Applications

The data available for mining interesting knoweldge (e.g., census data, corporate data, biological data) is often in a bad shape (have been gathered under no particular considerations), e.g. it may contain dublicate or incomprehensible information. Therefore a preprocessing stage is required to clean the data. The main data mining tasks are considered to be those that have an almost well defined algorithmic objective and assume that the given data are in a good shape. Moreover after the results of a data mining task are obtained they may need a post processing stage to interprete and vizualize them.

We give a list of data mining tasks:

- *Association rules:* Find correlations among the columns of the input matrix of the form: if there is a 1 entry in column 5 then most probably there is a 1 entry in column 7 too. These rules are probabilistic in nature.
- *Sequential patterns:* Find sequential patterns that occur often in a dataset.
- *Time series similarity:* Find criteria that check in a useful way whether two sequences of data exhibit "similar features".
- *Sequence matching:* Given a collection of sequences and a sequence query, find the sequence which is closest to the query-sequence.
- *Clustering:* Partition a given set of points into groups, called clusters so that "similar" points belong to the same cluster. A measure of similarity is needed, often it is a distance in a metric space or in a vector space.
- *Classification:* Given a set of points and a set of labels, assign labels to point so that similar objects are labeled by similar labels and a point is labeled by a more likely label. A measure of similarity of points and similarity of labels is assumed and a likelihood of a point to be assigned a particular label.

- *Discovery of outliers:* Discover points in the dataset which are isolated, i.e., they do not belong to any multi-populated cluster.
- *Frequent episodes:* An extension of sequential pattern finding, where more complex patterns are considered.

These tasks as described in the list are usually decomposed in more primitive modules that may be common in several tasks, e.g., comparing large pieces of the input matrix to find similarity patterns is useful in clustering, and association rules mining.

We include a list of some of the most common applications of data mining:

- *Marketing.* One of the most known successes of data mining. Market basket mining is motivated by the decision support problem which is based on basket data collected by most large retail organizations and used mostly for marketing purposes. In this context, we care for discovering association rules, e.g., if a person buys pencil then most probably buys paper too. Use of association rules resulted in observing customer habits to decide placement of objects in a supermarket so as to increase sales or decide on the prices of plane tickets.
- *Astronomy.* Clustering celestial objects by their radiation to distinguish galaxies and other star formations.
- *Biology.* Correlate diabetes to the presence of certain genes. Find DNA sequences representing genomes (sequential patterns).
- *Document analysis.* Cluster documents by subject. Used, for example, in collabnorative filtering, namely tracking user behavior and making recommendations to individuals based on similarity of their preferences to these of other users.
- *Financial Applications.* Use time series similarity to find stocks with similar price movements or find products with similar selling patterns. Observe similar patterns in customers' financial history to decide if a bank loan is awarded.
- *Web mining.* Search engines like Google rank web pages by their "importance" in order to decide the order on which to present search results on a user query. Identifying communities on the web, i.e., groups that share a common interest and have a large intersection of web pages that are most often visited by the members of a group. This may be useful for advertising or to identify the most up-to-date information on a topic or to provide a measure of page rank which is not easy to spam. One popular method is to study co-citation and linkage statistics: web communities are characterized by dense directed bipartite subgraphs.
- *Communications.* Discover the geographic distribution of cell phone traffic at different base stations or the evolution of traffic at Internet routers over time. Detecting similarity patterns over such data is important, e.g., which geographic regions have similar cell phone usage distribution, or which IP subnet traffic distributions over time intervals are similar.
- *Detecting intrusions.* Detecting intrusions the moment they happenn is important to protecting a network from attack. Clustering is a technique used to detecting intrusions.
- *Detecting failures in network.* Mining episodes helps to detect faults in electricity network before they occur or detect congestions in packet switched networks.

4 Computing an approximation of the L_p distance

We consider in this section the following problem which may be part of various data mining tasks. The data stream model is assumed and we want to compute an approximation to the L_p distance. Formally, we are given a stream S of data items. Each data item is viewed as a pair (i, v), $i = 1, \ldots, n$, with entries for v an integer in the range $\{-M, M\}$ where M is a positive integer (so we need $\log M$ memory to store the value of each data item). Note that there may exist several pairs (with possibly different values for v) for a specific i. We want to compute a good approximation of the following quantity:

$$L_p(S) = (\Sigma_{i=1,\ldots,n} |\Sigma_{(i,v)\in S} v|^p)^{1/p}$$

The obvious solution to this problem, i.e., maintain a counter for each i is too costly because of the size of the data. In the influential paper [7], a scheme is proposed for approximating $L_2(S)$ in workspace $O(1/\epsilon)$ with arbitrarily large constant probability.

In [42], a solution for $L_1(S)$ is investigated for the special case where there are at most two non zero entries for each i. In this case, the problem can be equivalently viewed as having two streams S_a and S_b and asking for a good approximation of $L_1(S_a, S_b) = \Sigma_i |\Sigma_{(i,v)\in S_a} v - \Sigma_{(i,v) v \in S_b} v|$. A single pass algorithm is developed in [42], which, with probability $1 - \delta$, computes an approximation to $L_1(S)$ within a factor of ϵ using $O(\log M \log n \log(1/\delta)/\epsilon^2)$ random access space and $O(\log n \log\log n + \log M \log(1/\delta)/\epsilon^2)$ computation time per item. The method in [42] uses *sketches of vectors*, which can be viewed as a *summary data structure*. In this case, a sketch $C(S_a)$, $C(S_b)$ is computed for each data stream S_a, S_b respectively. Sketches are much smaller in size than S_a, S_b and such that an easily computable function of the sketches gives a good approximation to $L_1(S_a, S_b)$.

In [53], a unifying framework is proposed for approximating $L_1(S)$ and $L_2(S)$ within a factor of ϵ (with probability $1-\delta$) using $O(\log M \log(n/\delta) \log(1/\delta)/\epsilon^2)$ random access space and $O(\log(n/\delta))$ computation time per item. The technique used in [53] combines the use of stable distributions [74] with Nisan pseudorandom generators [73]. The property of stable distributions which is used in this algorithm is the following. The dot product of a vector $\vec{u}$ with a sequence of n independent identically distributed random variables having p-stable distribution is a good estimator of the L_p norm of $\vec{u}$. In particular we can use several such products to embed a d-dimensional space into some other space (of lower dimensionality) such that to approximately preserve the L_p distances. Dot product can be computed in small workspace.

We shall describe here in some detail the first stage of the algorithm in [53] for approximating $L_1(S)$: For $l = O(1/\epsilon^2 \log 1/\delta)$, we initialize nl independent random variables $X_i^j, i = 1, \ldots, n, j = 1, \ldots, l$ with Cauchy distribution defined by the density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$ (we know this distribution is 1-stable). Then, the following three steps are excuted:

1. Set $S^j = 0$, for $j = 1, \ldots, l$.
2. For each new pair (i, v) do: $S^j = S^j + vX_i^j$ for all $j = 1, \ldots, l$.

3. Return the median $(|S^0|, \ldots, |S^{l-1}|)$.

To prove the correctness of this algorithm we argue as follows: We want to compute $L_1(S) = C = \Sigma_i |c_i|$ where $c_i = \Sigma_{(i,v) \in S} v$. First, it follows from the 1-stability of the Cauchy distribution that, each S^j has the same distribution as CX where X has Cauchy distribution. Because X has Cauchy distribution, $median(|X|) = 1$ hence $median(v|X|) = v$ for any v. It is known that for any distribution, if we take $l = O(1/\epsilon^2 \log 1/\delta)$ independent samples and compute the median M, then the distribution function of M is within ϵ of $1/2$ with probability $> 1 - \delta$. Thus, it can be proven that $l = O(1/\epsilon^2 \log 1/\delta)$ independent samples from the Cauchy distribution approximate the median within a factor of ϵ with probability $> 1 - \delta$.

This stage of the algorithm, though, assumes random numbers of *exact* precision. Thus, random generators are used in [53]to tackle the problem of how to reduce the number of required random bits.

The problem of approximating L_p distances in one pass algorithms has a variety of other applications too, e.g., in estimating the size of self join [7, 45] and in the estimation of statistics of network flow data [42].

Computing sketches is a technique used with success to get summaries of the data. It has enabled compression of data and speeding up computation for various data mining tasks [54, 32, 23, 24, 33]. See also in section 5 for a description of the algorithm in [32].

For an example, in [33] sketches are used based on random projections to approximate a similarity measure depending on L_p distance. The sketches used (which are vectors of much lower dimensionality) approximately preserve distances and are defined as: The i-th component of the sketch $\vec{s}(\vec{x})$ of $\vec{x}$ is the dot product of $\vec{x}$ with a random vector $\vec{r}_i$.

$$\vec{s}_i(\vec{x}) = \vec{x} \cdot \vec{r}_i$$

where each component of each random vector is drawn from a Cauchy distribution.

5 Association Rules

One of the most successful applications of data mining is identifying association rules in market basket data. The problem of mining for association rules has been studied extensively and many efficient heuristics are known. We will mention some of them and also discuss considerations on analyzing the problem and finding the formalisms that are closer to practical concerns. For example, when the results are used by retail organizations for marketing purposes then it is of interest to discover rules that hold for a large number of customers, as in the diapers/beer example. It was noticed that most customers who buy diapers also buy beer so placing diapers close to beer in the supermarket resulted in an increase of sales. However in a different setting it might be also of interest that two items are highly correlated even if both occur in very few baskets.

Basket data is a collection of records (or baskets), each record typically consisting of a transaction date and a collection of items (thought of as the items bought in this transaction). The problem of mining association rules over basket data was introduced in [3]. Formally we consider a domain set $\mathcal{I} = \{i_1, \ldots, i_m\}$ of elements called *items* and we are given a set $\mathcal{D}$ of transactions where each transaction T is a subset of $\mathcal{I}$. We say that a transaction T contains a set X of items if $X \subseteq T$. An *association rule* is an "implication" rule $X \Rightarrow Y$ where $X \subset \mathcal{I}$ and $Y \subset \mathcal{I}$ and X, Y are disjoint. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if $c\%$ of the transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if $s\%$ of the transactions in $\mathcal{D}$ contain $Y \cup X$. The symbol $\Rightarrow$ used in an association rule is not a logical implication, it only denotes that the confidence and the support are estimated above the thresholds $c\%$ and $s\%$ respectively. In this framework, the problem of mining for association rules on a given transaction set asks to generate all association rules with confidence and support thresholds greater than two given integers.

Functional dependencies are association rules with 100% confidence and any support and they are denoted as $X \rightarrow A$. Consequently, having determined a dependency $X \rightarrow A$, any dependency of the form $X \cup Y \rightarrow A$ can be ignored as redundant. The general case of association rules however is probabilistic in nature. Hence a rule $X \Rightarrow A$ does not make rule $X \cup Y \Rightarrow A$ redundant because the latter might not have minimum support. Similarly, rules $X \Rightarrow A$ and $A \Rightarrow Z$ do not make rule $X \Rightarrow Z$ redundant because the latter might not have minimum confidence.

Each transaction may be viewed equivalently as a row in a $n \times k$ 0,1-matrix where 1 means that the item represented by this column is included in this transaction and 0 that it is not included. The rows represent the baskets and the columns represent the items in the domain. The columns are sometimes called attributes or literals.

However confidence and support are not the only measures of "interestingness" of an association rule and do not always capture the intuition. Confidence is measured as the conditional probability of X given Y and it ignores the comparison to the (unconditional) probability of X. If the probability of X is high then confidence might be measured above threshold although this would not imply any correlation among X and Y. Other measures considered are the *interest* and the *conviction* [19]. Interest is defined as the probability of both X and Y divided by the product of the probability of X times the probability of Y. It is symmetric with respect to X and Y, and measures their correlation (or how far they are from being statistically independent). However, it can not derive an implication rule (which is non-symmetric). Conviction is defined as a measure closer to the intuition of an implication rule $X \Rightarrow Y$. Its motivation comes from the observation that if $X \Rightarrow Y$ is viewed as a logical implication, then it can be equivalently written as $\neg(X \wedge \neg Y)$. Thus conviction measures how far from statistical independence are the facts X and $\neg Y$ and is defined as follows: $\frac{P(X)P(\neg Y)}{P(X, \neg Y)}$. In [18] conditional probability is not used to measure interestingness of an association rule and propose statistical correlation instead. In [81], *causal* rules instead of mere associations are discussed aiming to capture the intuition whether $X \Rightarrow Y$ means that X causes Y or some other item causes them both to happen. This direction of investigation is taken by noticing that yielding a small number (possibly arbitrarily decided) of the "most interesting" causal relationships is of interest, as data mining is used for exploratory analysis. In that perspective, it is pointed out that ongoing research in Bayesian learning (where several techniques are developed to extract causal relationships) seems promising for large scale data mining.

In the context of the association rule problem, mining for frequent itemsets is one of the major algorithmic concerns. The *frequent itemsets problem* asks to find all sets of items (*itemsets*) that have support above a given threshold. This problem can be reduced to finding all the *maximal frequent itemsets* due to the monotonicity property. A frequent itemset is maximal if any itemset which contains it is not frequent. Clearly any subset of a frequent itemset is also a frequent itemset. This is the monotonicity property on which the a-priori trick is based. Many of the known techniques are a-priori algorithms, namely they begin the search for frequent itemsets with searching for frequent items and then constructing candidate pairs of items only if both items in the pair are frequent. Thus, to find frequent itemsets, they proceed levelwise, finding first the frequent items (sets of size 1), then the frequent pairs, the frequent triples, and so on.

Mining for frequent itemsets

An *a-priori algorithm* [3, 5] which finds first all frequent items needs to store them in main memory (it assumes that there is enough space) and then create candidate sets of all pairs of frequent items and finds in a second pass all frequent pairs. After the second pass it can create candidate sets of triples such that any subset of a candidate set is a frequent pair, and so on. It needs so many passes through the data as the maximum size of a frequent itemset or two passes if we are only interested in frequent pairs as is the case in some applications. Improvements have been introduced in this original idea which address issues such as: if the main memory is not enough to accommodate counters for all pairs of items, then e.g., hashing is used to prune some infrequent pairs in the first pass.

In [19], the number of passes is reduced by taking a dynamic approach to the apriori algorithm which is called Dynamic Itemset Counting. It reduces the number of passes of apriori by starting counting 2-itemsets (and possibly 3-itemsets) during the first pass. After having read (say) one third of the data, it builds candidate 2-itemsets based on the frequent 1-itemsets count so far. Thus running on the rest two thirds of the data, it checks also the counts of these candidates and it stops checking the 2-itemsets counts during the second pass after having read the first third of data. Similarly, it may start considering 3-itemsets during the first pass after hiving read the first two thirds of the data and stops considering them during the second run. If the data is fairly homogeneous, this algorithm finds all frequent itemsets in around two passes.

In [78], a hash table is used to determine on the first pass (while the frequent items are being determined) that many pairs are not possibly frequent (assuming that there is enough main memory). The hash table is constructed so that each of its buckets stores the accumulative counts of more than one pairs. This algorithm works well when infrequent pairs have small counts so that even when all the counts of pairs in the same bucket are added, the result is still less than the threshold. In [40], multiple hash tables are used in the first pass and a candidate pair is required to be in a large bucket in *every* hash table. In the second pass another hash table is used to hash pairs and in the third pass, only if a pair belongs to a frequent bucket in pass two (and has passed the test of pass one too) is taken as a candidate pair. The multiple hash tables improve the algorithm when most of the buckets have counts a lot below the threshold (hence many buckets are likely to be small).

These methods, however, cannot be used for finding all frequent itemsets in one or two passes. Algorithms that find all frequent itemsets in one or two passes usually rely on randomness of data and sampling. A simple approach is to take a main-memory-sized sample of the data, run one of the main-memory algorithms, find the frequent itemsets and either stop or run a pass through the data to verify. Some frequent itemsets might be missed in this way. In [83] this simple approach is taken. The algorithm on main memory is run on a much lower threshold so it is unlikely that it will miss a frequent itemset. To verify, we add to the candidates of the sample the negative border: those sets of items such that each itemset S is not identified as frequent in the sample, but every immediate subset of S is. The candidate itemsets includes all itemsets in the negative border. Thus the final pass through the data counts the frequency of the itemsets in the negative border. If no itemset in the negative border is frequent, then the sample has given all the frequent itemsets candidates. Otherwise, we may rerun the whole procedure if we want not to miss any frequent itemset.

Mining for Similarity Rules

As pointed out, various other kinds of rules may be of interest given a set of basket data. A *similarity rule* $X \simeq Y$ denotes that the itemsets X and Y are highly correlated, namely they are contained both in a large fraction of the transactions that contain either X or Y. A similarity rule does not need to satisfy a threshold on the support, low-support rules are also of interest in this setting. Although for market basket analysis, the low support mining might not be very interesting, when the matrix represents the web graph, then similar web sites with low support might encompass similar subjects or mirror pages or plagiarism (in this case, rows will be sentences and columns web pages).

As low support rules are also sought for, techniques with support pruning (like finding all frequent itemsets) are not of use. In cases where the input matrix is sparse, however, and the number of columns is sufficiently small then we can store something per column in main memory. A family of algorithms were developed in [32] to solve the problem in those cases using a combination of random sampling and hashing techniques.

For each column C, a signature $S(C)$ is defined which, intuitively, is a summary of the column. Signatures are small enough such that a signature for each column can fit in main memory and, similar columns have similar signatures. As the matrix is sparse, we cannot choose a small number of rows at random and use each shortened column in this set of rows as the signature. Most likely almost all signatures will be all 0's. The idea in this paper is: For each pair of columns, ignore the rows that both columns have zero entries, find the fraction of rows that these columns differ (over all non-both-zero-entry rows) and define this as the similarity measure. Interestingly, it can be proven that this similarity measure is proportional to the probability that both rows have the first occurence of 1 in the same row. Thus the signature of each column is defined as the index of the first row with a 1 entry.

Two techniques that are developed in [32] are Min-Hashing (inspired by an idea in [31] –see also [22]) and Locality-Sensitive Hashing (inspired by ideas used in [49]–see also [55].

In Min-Hashing, columns are hashed to the same bucket if they agree on the index of the first row with a 1 entry. To reduce the probability of false positives and false negatives, a set of p signatures are collected instead of one signature. This is done by implicitly considering a set of p different random permutations of the rows and for each get a different hash function. For each column, we use as its new signature the sequence of the p row indices (the row where the first 1 entry appears in this column). Actually these p row indices can be derived using only one pass through the data by hashing each row using p different hash functions. However, if the number of columns is very large and we cannot afford work which is quadratic on the number of columns, then Locality-Sensitive Hashing is proposed. Locality-Sensitive Hashing aims at reducing the number of pairs of columns that are to be considered by finding quickly many non-similar pairs of columns (and hence eliminating those pairs from further consideration). Briefly, it works as follows: It views the signatures in each column as a column of integers. It partitions the rows of this collection of rows into a number of bands. For each band it hashes the columns into buckets. A pair of columns is a candidate pair if it hashes in the same bucket for all bands. However Locality-Sensitive Hashing (unlike Min-Hashing) has the disadvantage that by reducing the number of false negatives it increases the number of false positives and vice versa.

These algorithms, although very efficient in practice, might still yield *false positives* and *false negatives*, i.e., yield a similarity rule which is false or miss some similarity rules. In [43], a family of algorithms is proposed which is called *Dynamic Miss-Counting (DMC)* that avoid both false positives and false negatives. Two passes over the data are made and the amount of main memory used allows for data of moderate size. The key idea in DMC algorithms is confidence-pruning. For each pair of columns the algorithm counts the number of rows with entries in these columns that disagree and if the count exceeds a threshold they discard this similarity rule.

The algorithms we mentioned in this section are proven efficient with experiments. For detailed report on the results on those experiments the reader may refer to the specific publications cited. For an excellent detailed exposition of algorithms mentioned in this section see [84].

5.1 Transversals

We point out here the connection between maximal frequent itemsets and transversals [83, 69] which are defined as follows: A *hypergraph* is a 0-1 matrix with distinct rows. Each row can be viewed as an *hyperedge* and each column as an element. A *transversal* (a.k.a. *hitting set*) is a set of elements such that each hyperedge contains at least one element from the set. A transversal is *minimal* if no subset of it is a transversal.

Recall that a *frequent itemset* is a subset of the columns such that the number of rows with 1 entries in all those columns is above some support threshold. A *maximal* frequent itemset is a frequent itemset such that no superset is a frequent itemset. Given a support value, an itemset belongs to the negative border iff it is not a frequent itemset and all its subsets are frequent itemsets. The following proposition states the relationship between transversals and maximal frequent itemsets.

Proposition 5.1 *Let H_{Fr} be the hypergraph of the complements of all maximal frequent itemsets, and let H_{Bd^-} be the graph of all itemsets in the negative border. Then the following holds:*

1. The set of all minimal transversals of H_{Fr} is equal to the negative border.

2. The set of all minimal transversals of H_{Bd^-} is equal to the set of all maximal itemsets. □

It is not difficult to prove. A transversal T of H_{Fr} has the property: For each maximal frequent itemset S, the transversal T contains at least one attribute which is not included in this itemset S. Hence the transversal is not a maximal frequent itemset. Hence a minimal transversal belongs to the negative border.

For an example, suppose we have four attributes $\{A, B, C, D\}$ and let all maximal frequent itemsets be $\{\{A, B\}, \{A, C\}, \{D\}\}$, then the hypergraph of complements of those itemsets contains exactly the hyperedges $\{\{C, D\}, \{B, D\}, \{A, B, C\}\}$. All minimal transversals of this hypergraph are $\{\{C, B\}, \{C, D\}, \{A, D\}, \{D, B\}\}$ which is equal to the negative border.

This result is useful because the negative border can be found easier in general and then used to retrieve the maximal frequent itemsets.

Transversals have been studied for a long time and hence this connection is useful. In [69] this result is extended in more general framework for which finding maximal frequent itemsets is a subcase. A connection is shown among the three problems of computing maximal frequent itemsets, computing hypergraph transversals and learning monotone boolean functions. This approach as well as the approach taken in [4] has its roots in the use of *diagrams* of models in model theory (see e.g., [28]).

6 Clustering

There are many different variants of the clustering problem and literature in this field spans a large variety of application areas and formal contexts. Clustering has many applications besides data mining including statistical data analysis, compression, vector quantization. It has been formulated in various contexts such as machine learning, pattern recognition, optimization and statistics. Several efficient heuristics have been invented. In this section, we will review some recent one pass algorithms and mention some considerations on the quality of clustering.

Informally, the clustering problem is that of grouping together (clustering) similar data items. One approach is to view clustering as a density estimation problem. We assume that in addition to the observed variables for each data item, there is a hidden, unobserved variable indicating the "cluster membership". The data is assumed to be produced by a model with hidden cluster identifiers. A mixture weight $w_i(x)$ is assumed for each data item x to belong to a cluster i. The problem is estimating the parameters of each cluster C_i, $i = 1, \ldots, k$, assuming the number k of clusters is known. The clustering optimization problem is that of finding parameters for each C_i which maximize the likelihood of the clustering given the model.

Most conventional clustering algorithms require space $\Omega(n^2)$ and require random access to the data. Hence recently several heuristics have been proposed for scaling clustering algorithms. Algorithms for clustering usually fall in two large categories *k-median approach* algorithms and *hierarchical approach* algorithms.

6.1 The k-median approach

A common formulation of clustering is the *k-median* problem: Find k centers in a set of n points so as to minimize the sum of distances from data points to their closest cluster centers. Or, equivalently, to minimize the average distance from data points to their closest cluster centers. The assumptions taken by the classical k-median approach are:

1)each cluster can be effectively modeled by a spherical Gaussian distribution, 2) each data item is assigned to one cluster and 3) the weights are assumed equal.

In [50], a constant-factor approximation algorithm for the k-median problem is proposed which uses a single pass on the data stream model and requires workspace $\theta(n^{\epsilon})$. A related problem is the k-center problem (minimize the maximum radius of a cluster) which is investigated in [30] where a single pass algorithm which requires workspace $O(k)$ is presented. In [75], results from [50] are used to develop an algorithm that achieves dramatically better clustering quality than BIRCH although it takes longer to run.

In [16], a single pass algorithm is presented for points in the Euclidean space. The method used is based in identifying regions of the data that are compressible (*compression set*), other regions that must be maintained in memory (*retained set*) and a third kind of regions that can be completely discarded (*discard set*). The algorithm is evaluated by experiments. The discard set is set of points that are certain to belong to a specific cluster. They are discarded after they are used to compute the statistics of the cluster (such as the number of points, the sum of coordinates, the sum of squares of coordinates). The compression set is set of points that are close to each other so that it is certain that they will be assigned to the same cluster. They are replaced by their statistics (same as for the discard set). In the retained set remain the rest of the points that do not belong to either of the two other categories. The algorithm begins by storing a sample of points (the first points to be read) in main memory and running on them a main memory algorithm (such as k-means). A set of clusters is obtained which will be modified as more points are read into main memory and processed. Then it processes, in each subsequent stage, a main-memory full set of points briefly as follows. 1. Determine if a set of points is (a) sufficiently close to some cluster c_i and (b) unlikely for c_i to "move" far from these points (during subsequent stages) and another cluster come closer. A discard set is decided in this way and its statistics used to update the statistics of the cluster. 2. Cluster the rest of the points in main memory and if a cluster is very tight, then replace the corresponding set of points by its statistics; this is a compression set. 3. Consider merging compression sets.

For arbitrary metric spaces, a single pass algorithm is developed in [44] which uses a data structure like an R-tree to store clusters and uses summaries of data in a similar manner as in [16].

Regarding the question of finding similarity measures according to which to cluster objects, in [33], methods for determining similar regions in tabular data (given in a matrix) are developed. The proposed similarity measure is tested using the k-means algorithm to cluster tabular data. This measure of similarity is based on the L_p norm for various values for p (non-integral too). It is noticed that on synthetic data, when clustering uses as a distance measure either L_1 or L_2 norms, the quality of the clustering is poorer than when p is between 0.25 and 0.8. The explanation for this is that, for large p, more emphasis is put on the outlier values, whereas for small p the measure approaches the Hamming distance, i.e., it counts how many values are different. On real data, it is noticed that different values for p bring out different features of the data. It therefore, seems that p can be used as a useful parameter of the clustering algorithm: set p higher to show full details of the data set, reduce p to bring out unusual clusters in the data. In [33], they use sketches [53] to approximate the distances between subtables and reduce the computation.

6.2 The hierarchical approach

A hierarchical clustering is a nested sequence of partitions of the data points. It starts with placing each point in a separate cluster and merges clusters until it obtains either a desirable number of clusters (usually the case) or a certain quality of clustering.

The algorithm CURE [51] handles large datasets and assumes points in Euclidean space. CURE employs a combination of random sampling and partitioning. A random sample, drawn from the data set, is first partitioned and cluster summaries are stored in memory in a tree data structure. For each successive data point, the tree is traversed to find the closest cluster to it.

6.3 Clustering documents by Latent Semantic Indexing (LSI)

The use of vector space models for information retrieval purposes has been used as early as 1957. The application of SVD in information retrieval is proposed in [35] through the latent semantic indexing technique and it is proven a powerful approach for dimension reduction. The input matrix $\vec{X}$, is a document vs. terms/words matrix. It could be a 0,1-matrix or each entry could be the frequency of the term in this document. Matrix $\vec{X}$ is approximated according to SVD by $\vec{X}_k = \vec{U}_k \vec{T}_k \vec{V}_k^{Tr}$. The choice of k is an issue for investigation. Note that each entry of the matrix $\vec{X}_k$ does not correspond to a term any more, it corresponds to a weighted sum of term measures. The matrix V_k represents similarities among documents, e.g., given a document that the user is interested in more documents can be decided that are of interest to this user (even if they do not use the same terms). The matrix U_k displays similarities between terms, e.g., given a term, other related terms may be decided (such the term "car" is related to "automobile" and "vehicle"). The matrix X_k may be used for term-document associations, e.g., on a given term, extract documents that contain material related to this term.

In the elegant paper [62], a survey is presented of spectral methods –i.e., the use of eigenvectors and singular vectors of matrices–in document information retrieval and the application of SVD through the latent semantic indexing technique is explained and further bibliographic references are given.

6.4 How good are clustering algorithms?

In a clustering algorithm the objective is to find a good clustering but a good clustering is not formally defined. Intuitively the quality of a clustering is assessed by how much similar points are grouped in the same cluster. In [58] the question is posed: how good is the clustering which is produced by a clustering algorithm? As already discussed the k-median clustering may produce a very bad clustering in case the "hidden" clusters are far from spherical. E.g., imagine two clusters, one that is a sphere and a second one is formed at a certain distance around the sphere forming a ring. Naturally the k-median approach will fail to produce a good clustering in this example. A bicretiria measure is proposed for assessing the quality of the cluster. The dataset is represented as a graph with weights on the edges that represent the degree of similarity between the two vertices (high weight means high similarity). First a quantity which measures the relative minimum cut of a cluster is defined. It is called *expansion* and is defined as the weight of the minimum cut divided by the number of points in the smaller subset among the two that the cut partitions the cluster-graph. It seems however that it is more appropriate to give more importance to vertices with many similar other vertices than to vertices with few similar other vertices. Thus, the definition is extended to capture this observation and the *conductance* is defined where subsets of vertices are weighted to reflect their importance. Optimizing the conductance gives the right clustering in the sphere-ring example. However if we assume the conductance as the measure of quality, then imagine a situation where there are mostly clusters of very good quality and a few points that create clusters of poor quality. In this case the algorithm might create many smaller clusters of medium quality. A second criterion is considered in order to overcome this problem. This criterion is defined as the fraction of the total weight of edges that are not covered by any cluster.

This bicriterion optimization framework is used to measure the quality of several spectral algorithms. These algorithms, though have proven very good in practice, were hitherto lacking a formal analysis.

Other popular clustering algorithms are k-means [56], CLARANS [72], BIRCH [89], DBSCAN [38],In [2], they define clusters in euclidean space by DNF formulas and address performance issues for data mining applications. In [76], the drawbacks of random sampling in clustering algorithms (e.g., small clusters might be missed) are avoided by density biased sampling. The goal is to under-sample dense regions and over-sample sparse regions of the data. A memory efficient single pass algorithm is proposed that approximates density biased sampling. An excellent detailed exposition of algorithms in [51], [16] and [44] can be found in [84]. An excellent survey of the algorithms in [48, 2, 11, 13] is given in [41].

7 Mining the web

The challenge in mining the web for useful information is the huge size and unstructured placement of data. Search engines aim to search the web for a specific topic and give the most important web pages on this topic to the user. A considerable amount of research has been done on ranking web pages according to their importance.

In [20], the search engine Google is presented and explained. The search results of Google are ranked according to the page citation importance. Thus an algorithm is developed for searching the web and rank pages. The algorithm uses "random jumps" from a web page to another to search the web for pages that provide information on a given topic. Random is needed to avoid dead ends. In [61], authorities and hubs are proposed following the observation that authorities might not link to one another directly but there are hubs that link "collectively" to many authorities. Thus hubs and authorities have this mutually depending relationship: good hubs link to many authorities and good authorities are linked by many hubs. Work in [47, 66] has shown that the concepts of hubs and authorities is a fundamental structural feature of the web. The CLEVER system [27] builds on the algorithmic framework of hub and authorities.

Techniques in both [20] and [61] for measuring the importance of a web page use iterative procedures. In [62], it is explained how these techniques are essentially spectral methods. The algorithm in [20] can be thought of as seeking an eigenvector of a probability matrix. Also, the limiting vectors for hub and authority importance in [61], are eigenvectors of matrices defined by the link graph of the web. The connections between this work and earlier work on sociology and citation analysis [36] is also pointed out in [62].

A recent work for on-line computation of the importance of web pages is [1].

In [21], the web is searched for frequent itemsets by a method using features of the algorithm for dynamic itemset counting [19]. Instead of a single deterministic run, the algorithm in [21] runs continuously exploring more and more sites. In [17], the extraction of structured data is achieved from information offered by unstructured data on the web. The example used is to search for books in the web starting from a small sample of books from which a pattern is extracted. Based on the extracted patterns more books are retrieved in a iterative manner.

8 Evaluating the results of data mining

As we have seen, for many of the successful data mining algorithms there is no formal analysis as to whether the solution they produce is a good approximation to the problem at hand. Recently a considerable amount of research is focused in developing criteria for such an evaluation.

A line of research is focused in building models for practical situations (like the link structure of the web or a corpus of technical documents) against which to evaluate algorithms. Naturally, the models, in order to be realistic, are shown to display several of the relevant features that are measured in real situations (e.g., the distribution of the number of outgoing links from a web page). In [77], a model for documents is developed on which the LSI method is evaluated. In this model, each document is built out of a number of different topics (hidden from the retrieval algorithm). A document on a given topic is generated by repeating a number of times terms related to the topic according to a probability distribution over the terms. For any two different topics there is a technical condition on the distributions that keeps the topics "well-separated". Their main result is that on this model, the k-dimensional subspace produced by LSI defines with high probability very good clusters as intended by the hidden parameters of the model. In this paper, they also proposed that if the dimensionality after applying LSI is too large, then random projection can applied to reduce it and improve the results. They suggested to project on a random subspace of dimensionality $O(\epsilon^{-2} \log n)$ and then compute the SVD. Other papers with results suggesting methods for evaluating spectral algorithms include [9].

Models for the web graph are developed. The web can be viewed as a graph with each page being a vertex and an edge exists if there is a hyperlink pointing from one web page to another. It has been observed that this graph has several characteristic features (work with measurements on the web graph include [65, 66]) which play an major role in the efficiency of several known algorithms for searching the web. In that context, the web graph is a power-law graph, which means roughly that the probability that a degree is larger than d is at least $d^{-\beta}$ for some $\beta > 0$. Models for power-law graphs are developed. In [39] a model is proposed of the internet growth and prove that under very general assumptions, it results in power-law-distributed degrees. Another work that proposes a random graph model for power-law graphs is [6]. A random graph model is proposed in [67].

In [63] they present a framework for evaluating the results of data mining operations according to the utility of the results in decision making which is formalized as an economically motivated optimization problem. This framework leads to interesting optimization problems such as the segmentation problem which is studied in [64]. Segmentation problems are related to clustering.

9 Conclusion

We discussed formal methods used for developing algorithms suitable for data mining applications where data are read into main memory only once and a limited amount of storage space is available. Then we discussed problems that are addressed in the data mining research and mentioned some related work done on those problems recently.

Among topics that are not mentioned are the following.

Work done on Time Series

Privacy preserving data mining

Semistructured data-XML Work done on Time Series

Bioinformatics

References

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *VLDB*, 2002.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in massive databases. In *SIGMOD*, pages 207–216, 1993.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. 1996.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[6] W. Aiello, F. Chung, and L. Lu. A random graph model for power law graphs. In *STOC*, 2000.

[7] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating frequency moments. In *STOC*, pages 20–29, 1996.

[8] N. Alon and J. Spencer. *The probabilistic method*. John Wiley and Sons, New York, 1992.

[9] Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *STOC*, pages 619–636, 2001.

[10] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.

[11] J. Banfield and A. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49:803–821, 1993.

[12] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *STOC*, 2001.

[13] A. Ben-Dor and Z. Yakhini. Clustering gene expression patterns. In *RECOMB*, 1999.

[14] J. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag, 1985.

[15] A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. In *STOC*, 1999.
[16] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *KDD*, 1998.
[17] S. Brin. Extracting patterns and relations from the world-wide web., 1998.
[18] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.
[19] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*, pages 255–264, 1997.
[20] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7/Computer Networks*, pages 107–117, 1998.
[21] S. Brin and L. Page. Dynamic data mining: Exploring large rule space by sampling., 1998.
[22] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, pages 21–29, 1997.
[23] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, 1998.
[24] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In *Sixth International World Wide We Conference*, pages 391–404, 1997.
[25] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. available at: http://www.cwi.nl/ rdewolf, 1999.
[26] R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53:17–25, 1995.
[27] S. Chakrabarti, B. Dom, R. Kumar, S. R. P. Raghavan, and A. Tomkins. Experiments in topic distillation. In *SIGIR workshop on hypertext information retrieval*, 1998.
[28] C. Chang and H. Keisler. *Model Theory*. North Holland, Amsterdam, 1990.
[29] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, pages 268–279, 2000.
[30] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *STOC*, pages 626–635, 1997.
[31] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and Systems Sciences*, 55:441–453, 1997.
[32] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. In *TKDE 13(1) 2001 and also in ICDE*, pages 64–78, 2000.
[33] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, 2002.
[34] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. In *FOCS*, pages 142–149, 1995.
[35] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *The American Society for Information Science*, 41(6):391–407, 1990.
[36] L. Egghe and R. Rousseau. *Introduction to Informetrics*. 1990.
[37] L. Engebretsen, P. Indyk, and R. O'Donnell. Derandomized dimensionality reduction with applications. In *SODA*, 2002.
[38] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knoweledge Discovery and Data Mining*, page 226, 1996.
[39] A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou. Heuristically optimized trade-offs: A new paradigm for power laws in the internet. In *STOC*, 2002.

[40] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. In *VLDB*, 1998.
[41] D. Fasulo. An analysis of recent work on approximation algorithms. Technical Report 01-03-02, University of Washington, Dept. of Computer science and Engineering, 1999.
[42] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l1-difference for massive data streams. In *FOCS*, 1999.
[43] S. Fujiwara, J. D. Ullman, and R. Motwani. Dynamic miss-counting algorithms: Finding implication and similarity rules with confidence pruning. In *ICDE*, pages 501–511, 2000.
[44] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French. Clustering large datasets in arbitrary metric spaces. In *ICDE*, pages 502–511, 1999.
[45] P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *SODA*, pages S909–S910, 1999.
[46] P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 281–291, 2001.
[47] D. Gibson, , and P. R. Jon M. Kleinberg. Inferring web communities from link topology. In *ACM Conference on Hypertext and Hypermedia*, volume 8(3-4), 1998.
[48] D. Gibson, J. M. Kleinberg, and P. Raghavan. Two algorithms for nearest neighbor search in high dimensions. In *STOC*, volume 8(3-4), 1997.
[49] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
[50] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, 2000.
[51] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD*, 1998.
[52] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. *Computing on data streams.* available at: http://www.research.digital.com/SRC/, 1998.
[53] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.
[54] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000.
[55] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.
[56] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
[57] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.
[58] R. Kannan, S. Vempala, and A. Vetta. On clusterings - good, bad and spectral. In *FOCS*, pages 367–377, 2000.
[59] A. Karlin, M. Manasse, L. Rodolph, and D. Sleator. Competitive snoopy caching. In *STOC*, pages 70–119, 1988.
[60] M. Kearns and U. Vazirani. *An introduction to comoputational learning theory.* MIT Press, 1994.
[61] J. Kleinberg. Authoritative sources in a hyperlinked environment. *J.ACM*, 46(5):604–632, 1999.
[62] J. Kleinberg and A. Tomkins. Applications of linear algebra in information retrieval and hypertext analysis. In *PODS*, pages 185–193, 1999.
[63] J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4):311–324, 1998.
[64] J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. Segmentation problems. In *STOC*, pages 473–482, 1998.

[65] S. Kumar, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *International World Wide Web Conference*, pages 309–320, 2000.
[66] S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling emerging cybercommunities automatically. In *International World Wide Web Conference*, volume 8(3-4), 1999.
[67] S. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *FOCS*, pages 57–65, 2000.
[68] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.
[69] H. Mannila and H. Toivonen. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems, Volume II, The Thirteenth European Meeting on Cybernetics and Systems Research*, pages 973 – 978, 1996.
[70] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
[71] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
[72] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.
[73] N. Nisan. Pseudorandom generators for pseudorandom computations. In *STOC*, pages 204–212, 1990.
[74] J. Nolan. *An introduction to stable distributions.* http://www.cas.american.edu/ jpnolan/chap1.ps.
[75] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *ICDE*, 2002.
[76] C. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. In *SIGMOD*, pages 82–92, 2000.
[77] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. *JCSS*, 61(2):217–235, 2000.
[78] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD*, pages 175–186, 1995.
[79] M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. In *STOC*, 2002.
[80] L. Schulman and V. Vazirani. Majorizing estimators and the approximation of ♯p-complete problems. In *STOC*, pages 288–294, 1999.
[81] C. Silverstein, S. Brin, R. Motwani, and J. D. Ullman. Scalable techniques for mining causal structures. In *Data Mining and Knowledge Discovery 4(2/3)*, pages 163–192, 2000.
[82] STREAM. Stanford stream data management project. http://www-db.stanford.edu/stream.
[83] H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145, 1996.
[84] J. Ullman. Lecture notes on data mining. available at: http://www-db.stanford.edu/˜ ullman/mining/mining.html, 2000.
[85] V. Vapnik. *Statistical learning theory.* John Wiley, 1998.
[86] V. Vazirani. *Approximation algorithms.* Springer, 2001.
[87] D. Vengroff and J. Vitter. I/o efficient algorithms and environments. *Computing Surveys*, page 212, 1996.
[88] J. Vitter. Random sampling with a reservoir. *ACM Trans. on Mathematical Software*, 11(1):37–57, 1985.
[89] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.